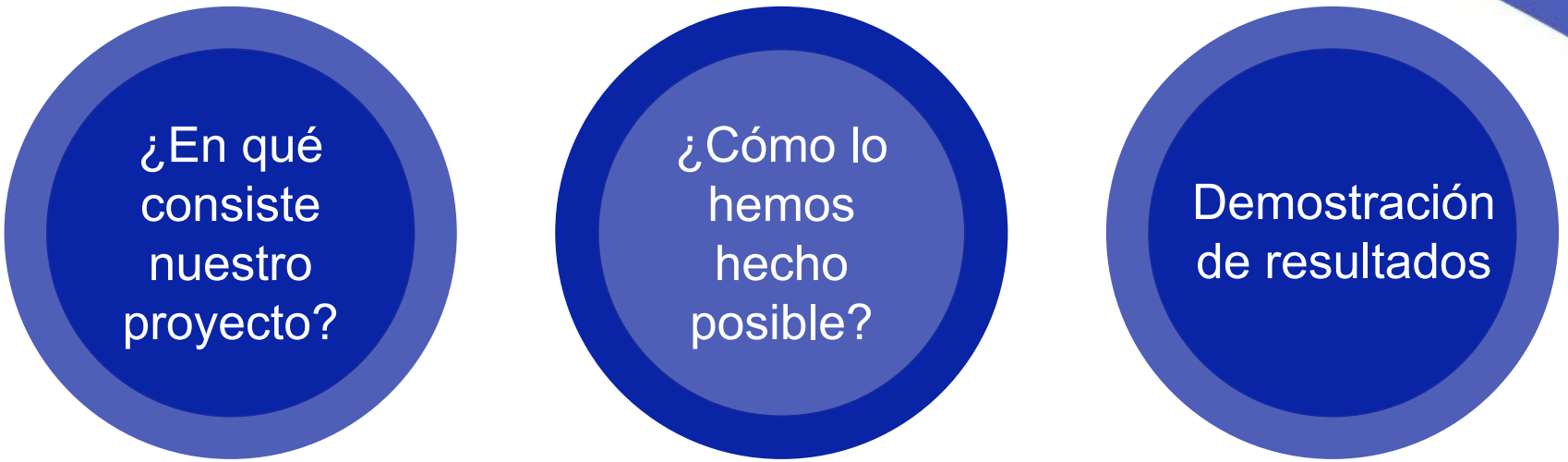
SAMSUNG

Identificador de comunidades tóxicas en las RRSS

Purple Team

Big Data Course

¿Qué os venimos a exponer?

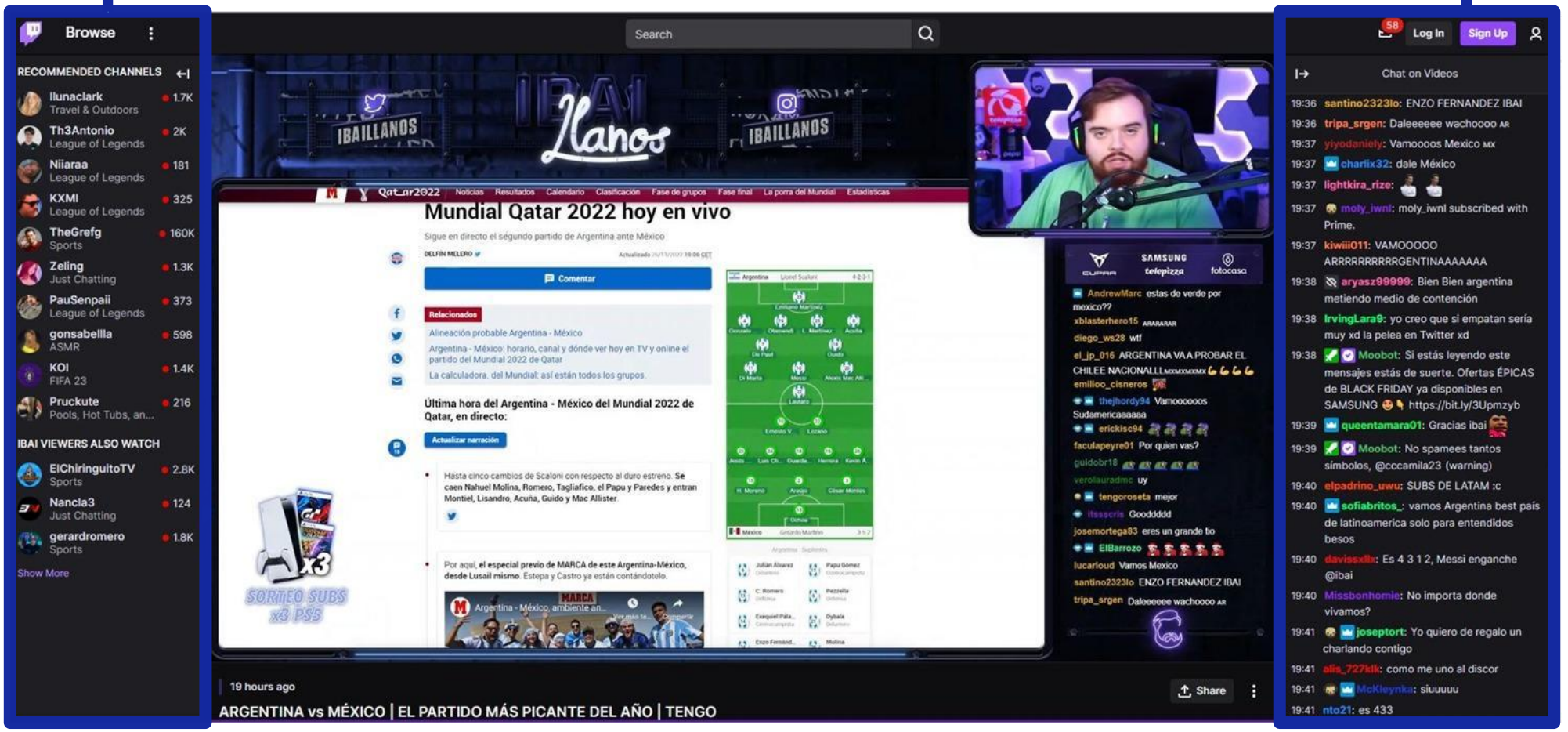


*"Creemos en la posibilidad de **construir comunidades dentro de internet más sanas**,*
*comunidades donde la **conversación** entre sus miembros es **asertiva**"*

¿En qué consiste nuestro proyecto?

Canales

Chat en directo

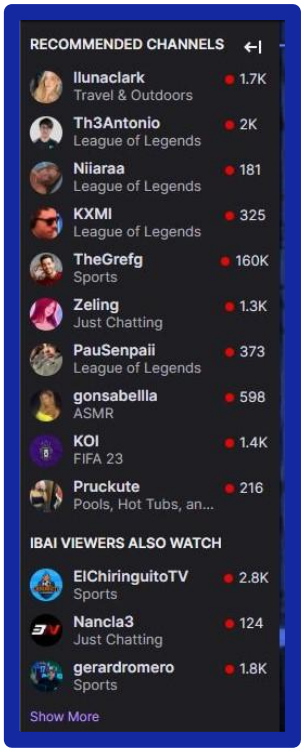


¿En qué consiste nuestro proyecto?

*"Implementación en Twitch de la posibilidad de conocer el **grado de asertividad de la conversaciones** que se llevan a cabo durante los directos de nuestros streamers favoritos"*

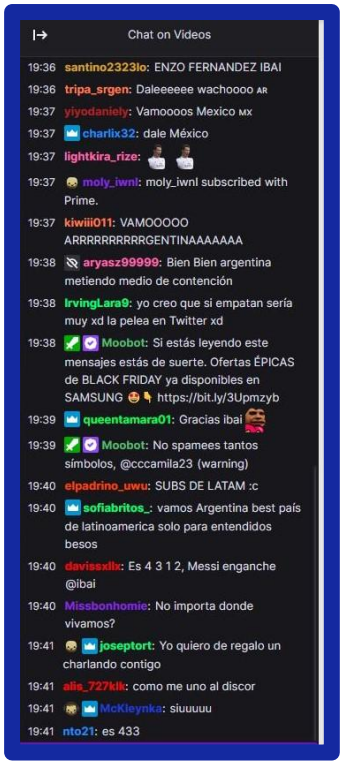
> 40 millones de visitantes por día

> 8 millones de creadores de contenido

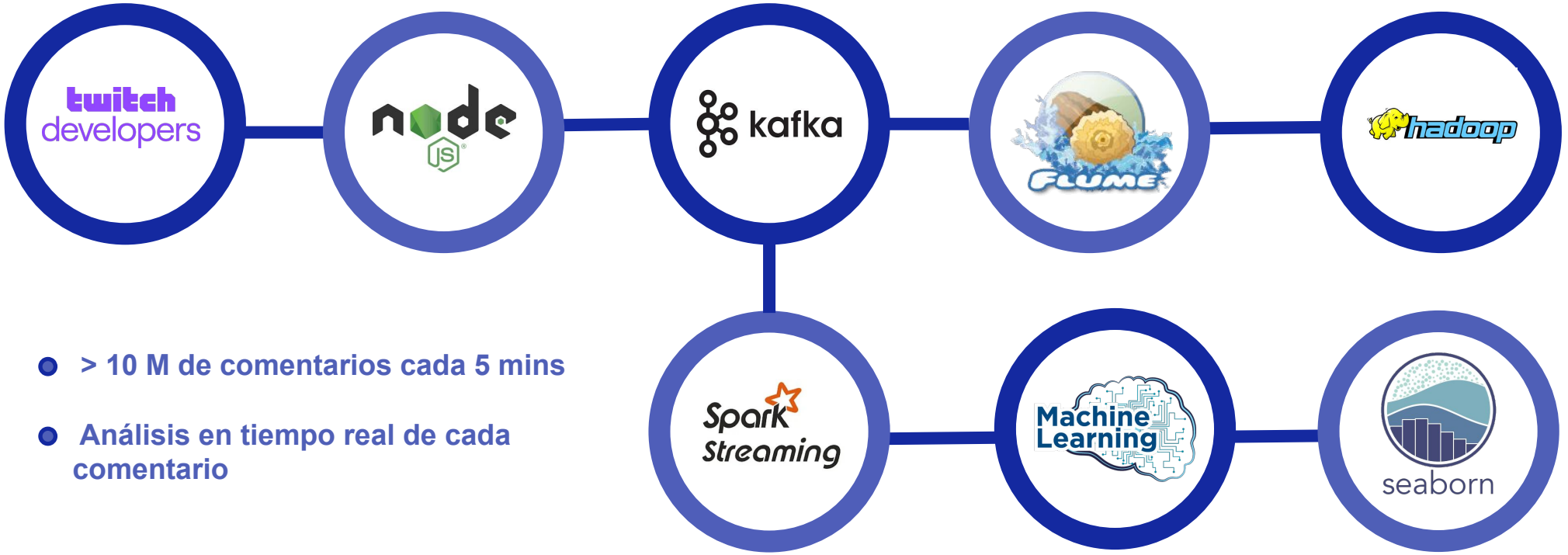


Extracción y filtrado de comentarios

Machine Learning para análisis y clasificación (0) (4)

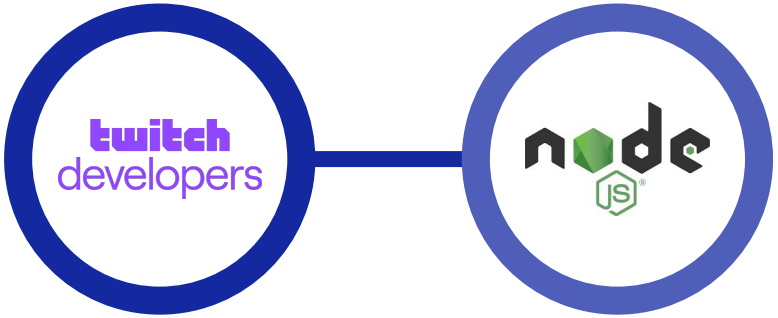


¿Cómo lo hemos hecho posible?



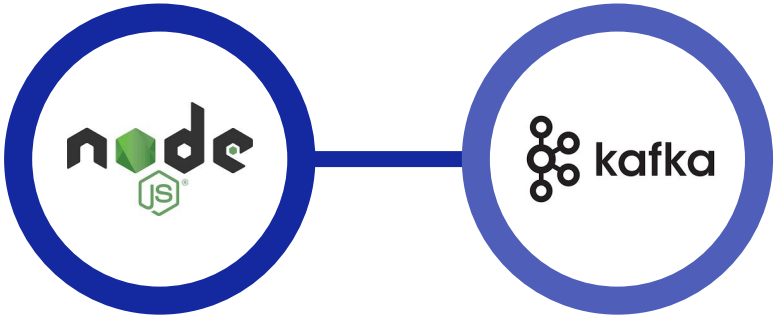
- **> 10 M de comentarios cada 5 mins**
- **Análisis en tiempo real de cada comentario**

1. Recolección de mensajes de chat en streaming



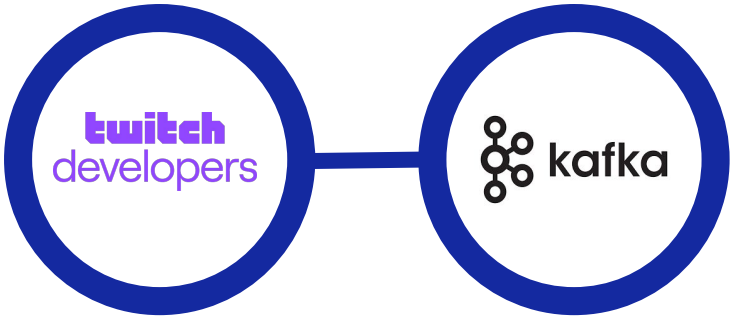

```
const client = new tmi.Client({
  connection: {
    secure: true,
    reconnect: true,
  },
  channels: channels[0],
});
client.connect();

client.on('message', (channel, tags, message, self) => {
```

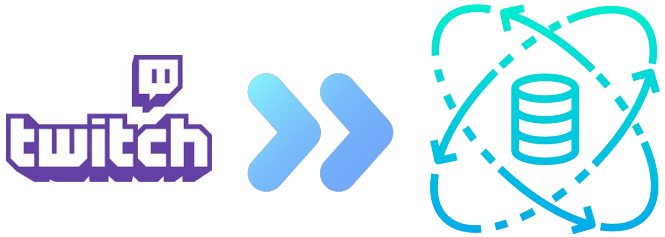
2. Implementación de Kafka con NodeJS



```
const kafkaClient = new kafka.KafkaClient({ kafkaHost: 'localhost:9092' });
const kafkaProducer = new kafka.HighLevelProducer(kafkaClient);

kafkaProducer.on('ready', () => {
  startClient(kafkaProducer);
});
```

3. Envío de mensajes en streaming

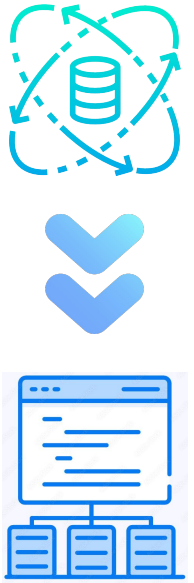




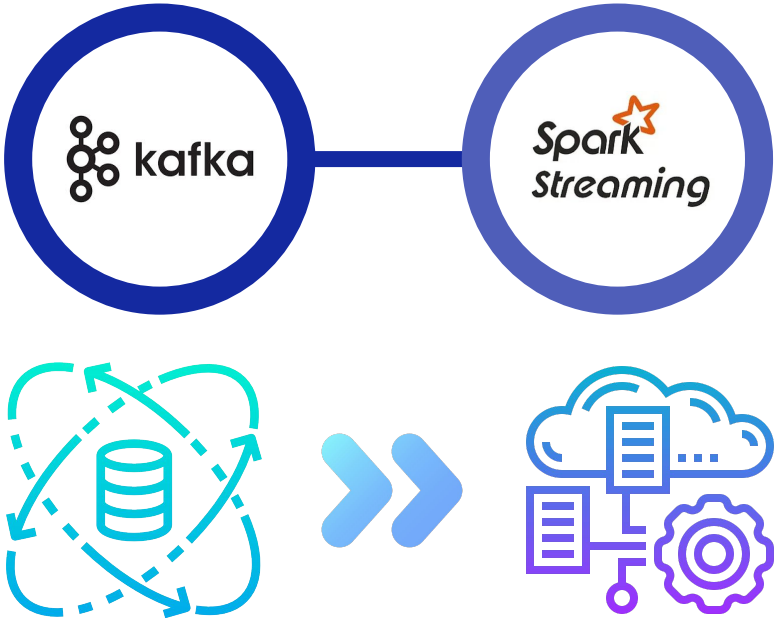

```
const startClient = async (producer) => {
  await getChannels();

  const client = new tmi.Client({···
  });
  client.connect();

  client.on('message', (channel, tags, message, self) => {
    if (tags.emotes == null && tags['message-type'] == 'chat' && !tags.mod) {
      payloads = [{
        topic: 'sparkTopic',
        messages: message,
        timestamp: Date.now(),
      },
      {
        topic: 'hdfsTopic',
        messages: message,
        timestamp: Date.now(),
      }];

      producer.send(payloads, (err, data) => {
        console.log(`DATA: ${JSON.stringify(data)}`);
      });
    }
  });
};
```

4. Almacenamiento de mensajes en Hadoop


```
agent1.channels = kafkaChannel
agent1.sinks = hdfsSink

agent1.sinks.hdfsSink.type = hdfs
agent1.sinks.hdfsSink.channel = kafkaChannel
agent1.sinks.hdfsSink.hdfs.writeFormat = Text
agent1.sinks.hdfsSink.hdfs.fileType = CompressedStream
agent1.sinks.hdfsSink.hdfs.codeC = snappy
agent1.sinks.hdfsSink.hdfs.batchSize = 10000
agent1.sinks.hdfsSink.hdfs.path = /user/student/twitch-messages/%Y-%m-%d/%H%M/%S
agent1.sinks.hdfsSink.hdfs.filePrefix = events-
agent1.sinks.hdfsSink.hdfs.round = true
agent1.sinks.hdfsSink.hdfs.roundValue = 10
agent1.sinks.hdfsSink.hdfs.roundUnit = minute
agent1.sinks.hdfsSink.hdfs.useLocalTimeStamp = true

agent1.channels.kafkaChannel.type = org.apache.flume.channel.kafka.KafkaChannel
agent1.channels.kafkaChannel.kafka.bootstrap.servers = localhost:9092
agent1.channels.kafkaChannel.kafka.topic = hdfsTopic
agent1.channels.kafkaChannel.parseAsFlumeEvent = false
```



5. Recepción de mensajes con Spark Streaming



In [ ]:
```
streamingRawDF = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "localhost:9092") \
  .option("subscribe", "sparkTopic") \
  .load()

streamingDF = streamingRawDF.selectExpr("CAST(value AS STRING) as text", "timestamp")
```

6.1. Creación del Modelo de Machine Learning


```
In [ ]: trainingDFAllCol = spark.read.format("csv") \
            .option("sep", ",") \
            .option("inferSchema", "true") \
            .load("training.csv")

        trainingDFPre = trainingDFAllCol.selectExpr("_c0 as label", "_c5 as text")
        trainingDF = trainingDFPre.withColumn("label", trainingDFPre["label"].cast("float"))
```

```
In [ ]: trainingDFSplit = trainingDF.randomSplit([0.8,0.2],2)
```

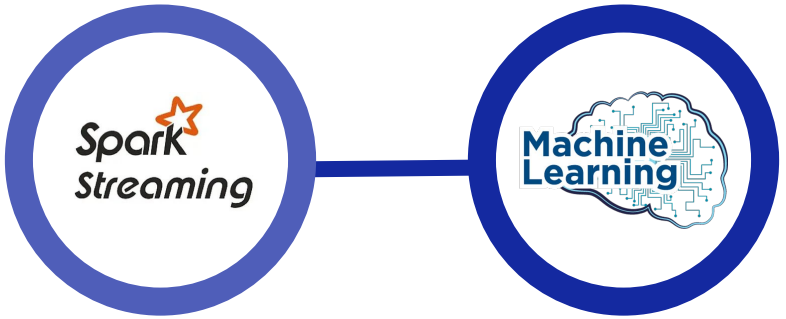
```
In [ ]: # Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
        tokenizer = Tokenizer(inputCol="text", outputCol="words")
        hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
        lr = LogisticRegression(maxIter=10, regParam=0.3, labelCol="label") #default regParam=0.001
        rfc = RandomForestClassifier(featuresCol="features", labelCol="label", numTrees=128, seed=42)
        dt = DecisionTreeClassifier(maxDepth=2, featuresCol="features", labelCol="label")
        pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

6.2. Entrenamiento y Test del Modelo de ML


```
In [ ]: model = pipeline.fit(trainingDFSplit[0])

In [ ]: model.save("file:/home/student/Desktop/twitch-big-data-project/models/lr_regParam0.3")

In [ ]: # Make predictions on test documents and print columns of interest
        prediction = model.transform(trainingDFSplit[1])
        predictionAndLabels = prediction.select("prediction", "label").rdd
        metrics = BinaryClassificationMetrics(predictionAndLabels)
        print("Area under ROC = %s" % metrics.areaUnderROC)
```

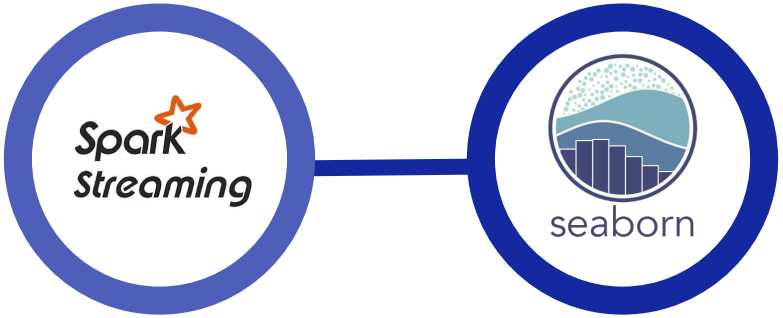
4 0

7. Aplicación del Modelo en tiempo real



```
In [ ]: lrModel = PipelineModel.load("file:/home/student/Desktop/twitch-big-data-project/models/lr_regParam0.3")

In [ ]: streamingPredictionDF = lrModel.transform(streamingDF).select('text', 'prediction', 'timestamp')
```

8. Configuración del flow de mensajes para Seaborn



```
In [ ]: df_count = (
            df_stream \
            .withWatermark("timestamp", "10 minutes") \
            .groupBy(window(col("timestamp"), "10 minutes", "10 minutes"), col("prediction")) \
            .count())
```

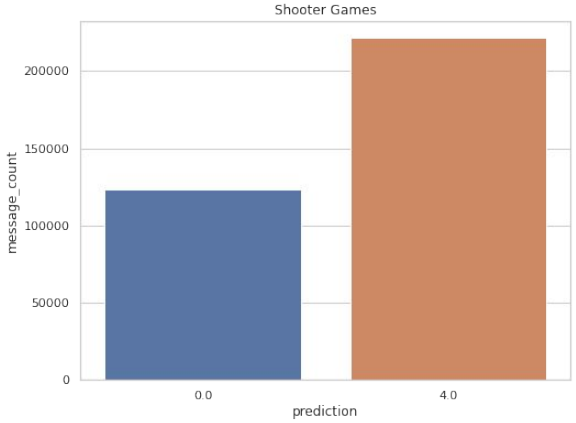
```
In [ ]: queryStream = (df_count \
         .writeStream \
         .format("memory") \
         .queryName("msg_changes") \
         .outputMode("update") \
         .start())
```

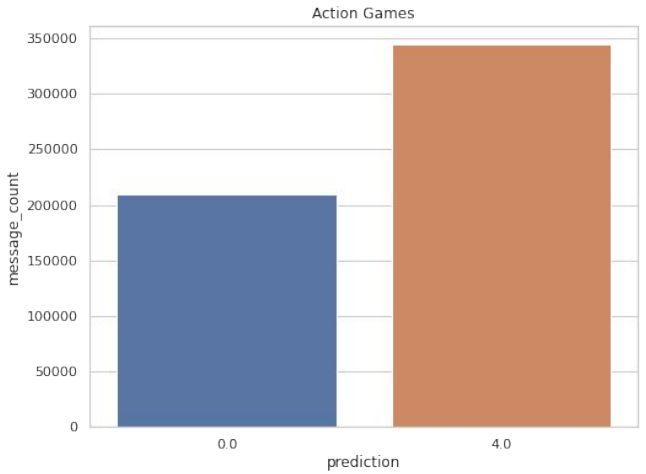
9. Visualización de los Datos en Streaming

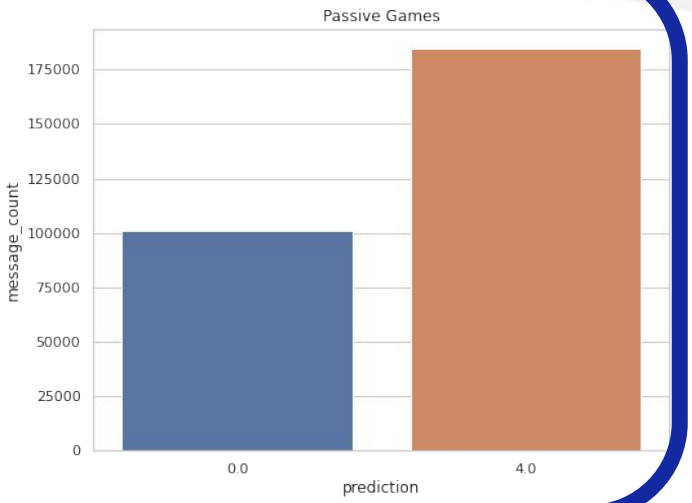
```
while True:
    # Clear output
    clear_output(wait=True)
    df = spark.sql(
            """
                select
                    window.start
                    ,window.end
                    ,prediction
                    ,sum(count) message_count
                from
                    msg_changes
                where
                    window.start = (select max(window.start) from msg_changes)
                group by
                    window.start
                    ,window.end
                    ,prediction
                order by
                    prediction desc
            """
    ).toPandas()

    sns.set_color_codes("muted")

    display(df)

    plt.figure(figsize=(8,6))
    try:
        # Barplot
        sns.barplot(x="prediction", y="message_count", data=df).set(title='Just Chatting')
        fig = plt.gcf()
        # Show barplot
        plt.show()
        sleep(10)
```

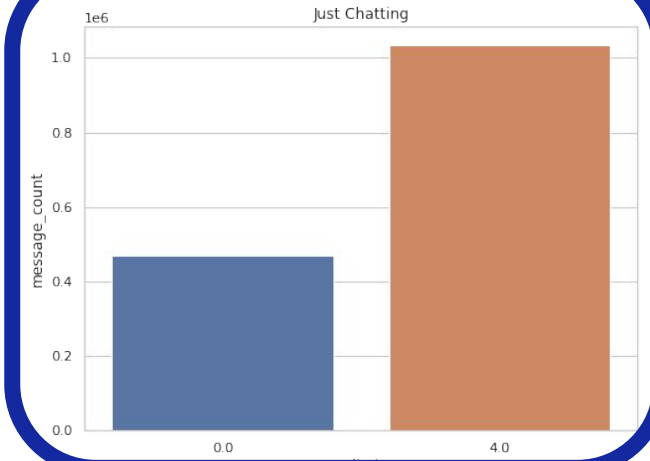
Demostración de resultados

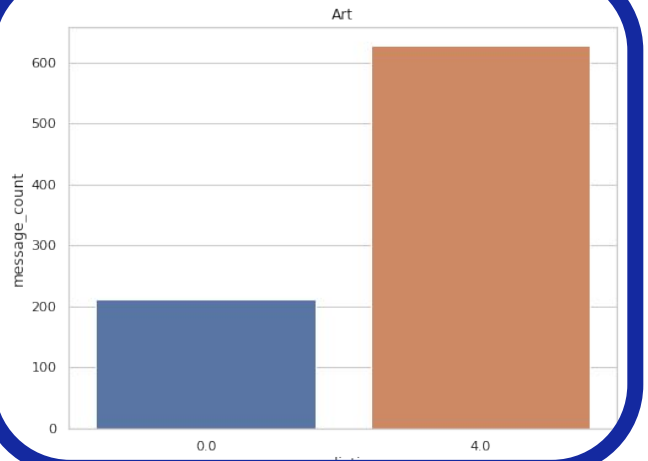
GAMES

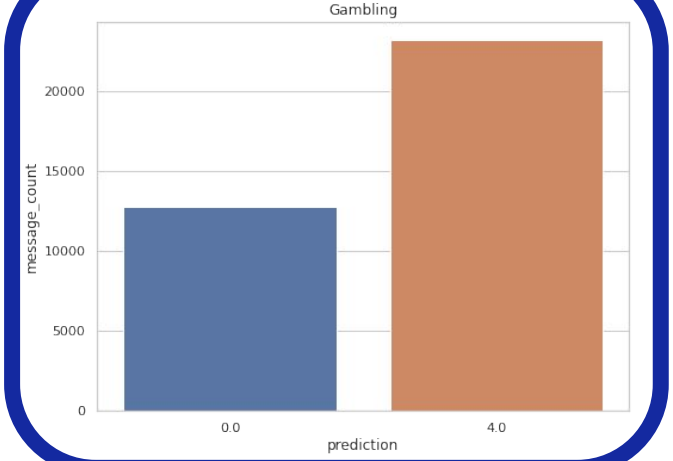




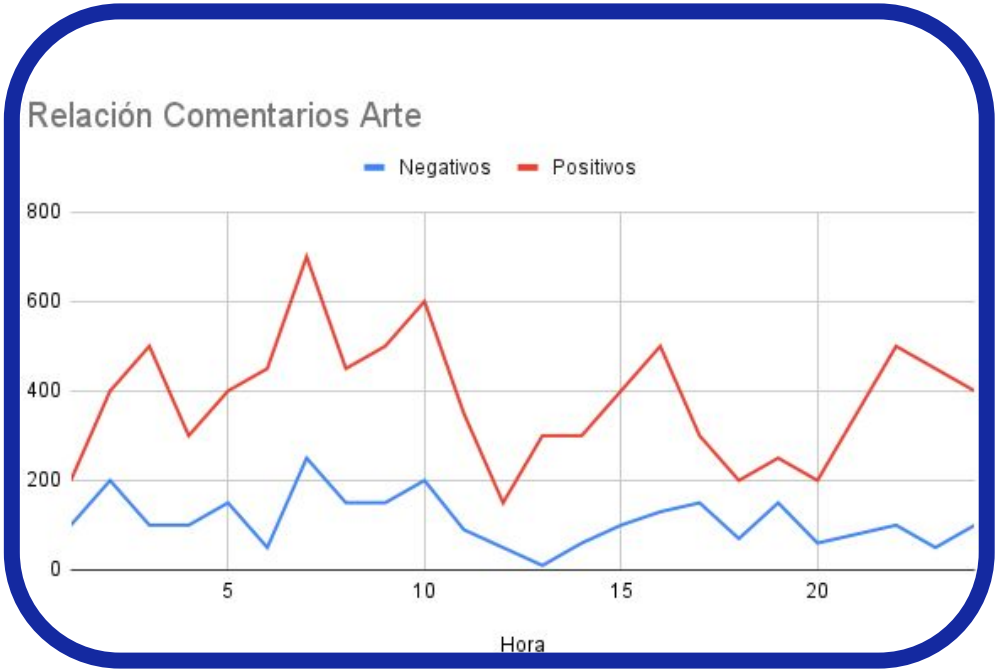




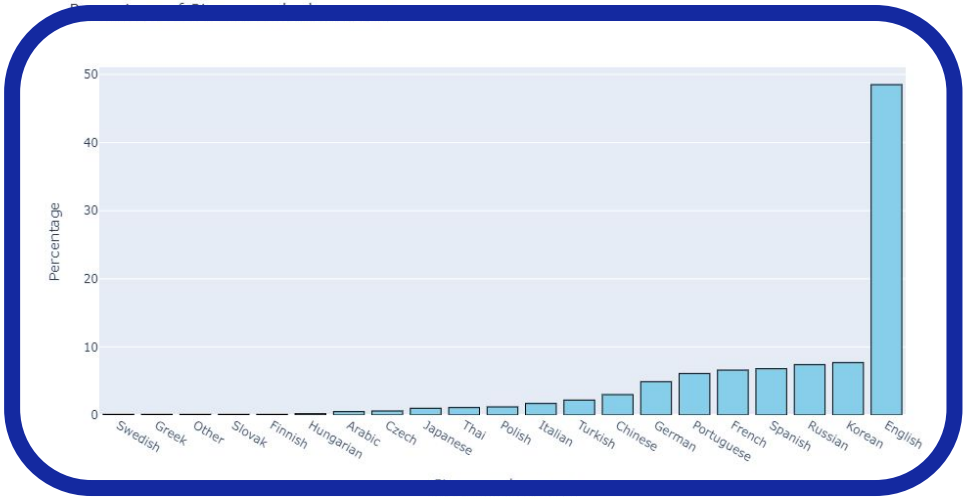




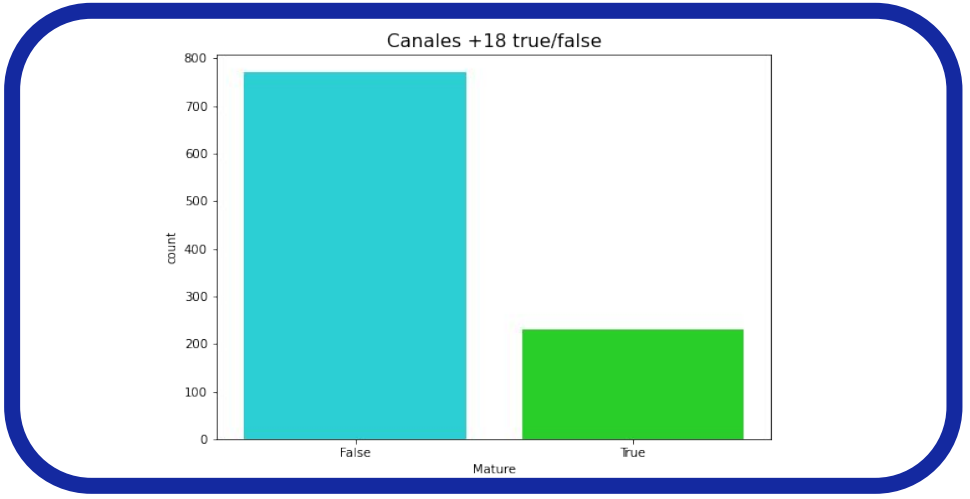
Demostración de resultados



Demostración de resultados

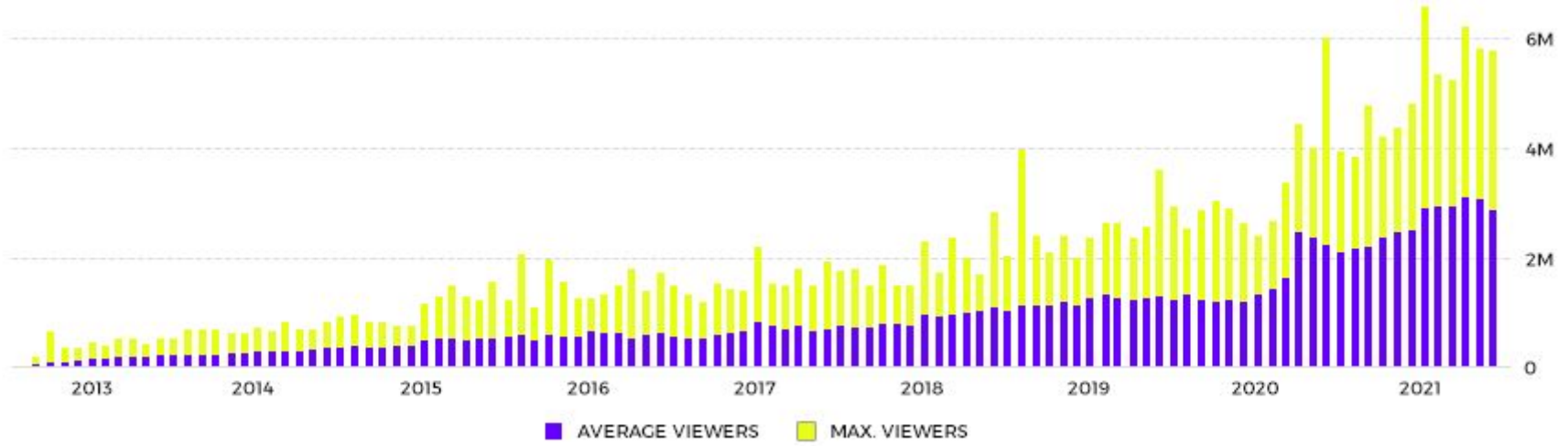


% de Streamers por lenguaje



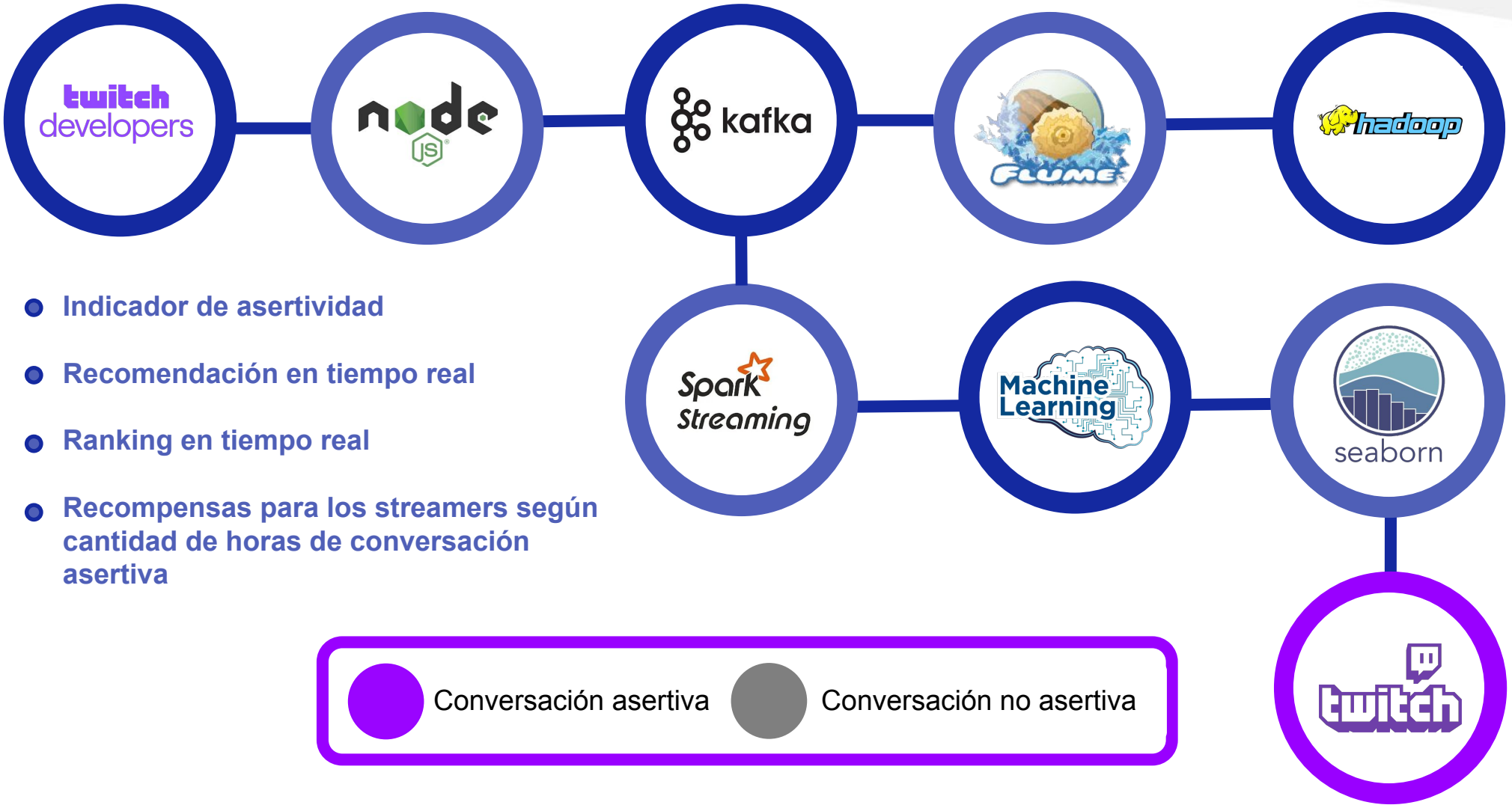
Categorización de canales en +18

Demostración de resultados



Crecimiento de usuarios en TWITCH

¿Qué viene ahora?



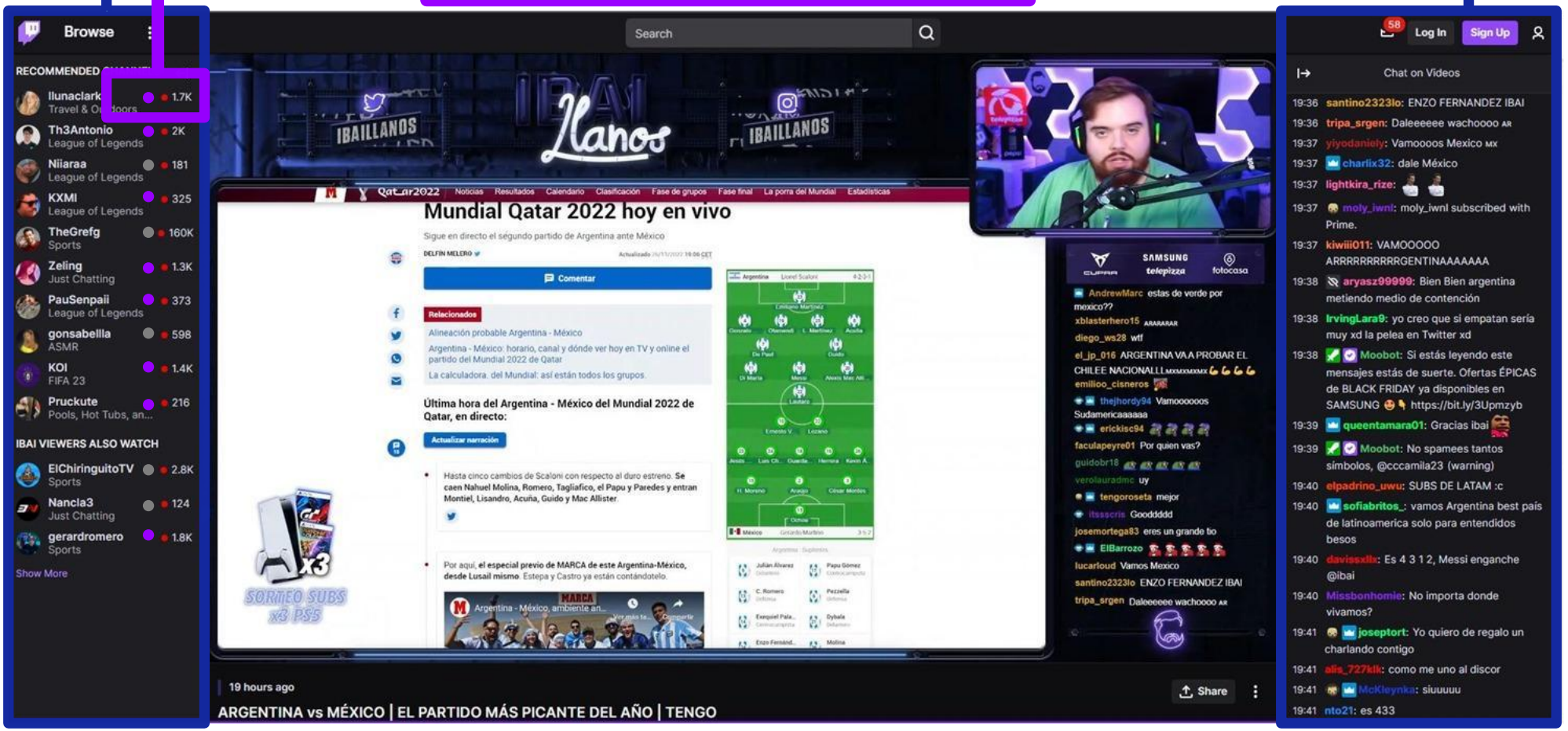
- **Indicador de asertividad**
- **Recomendación en tiempo real**
- **Ranking en tiempo real**
- **Recompensas para los streamers según cantidad de horas de conversación asertiva**

¿Qué viene ahora?

Recomendación de streamers

● Conversación asertiva ● Conversación no asertiva

Chat en directo



Queremos escucharos

¿Hemos despertado vuestra curiosidad?

¿Os parece realmente que podría servir de ayuda?